Sistema para reforzar la presencia de pequeñas PYME en Internet y fomentar la demanda en ámbito local de bienes de consumo.

Francisco Sánchez Gálvez

Resumen—Es una realidad que el consumidor está habituado a buscar y comparar bienes de consumo de diferentes proveedores mediante Internet. Actualmente la mayoría de los consumidores se desplazan a unos grandes almacenes o cadenas generalistas, que le ofrezcan la mejor calidad-precio, para adquirir aquellos productos que necesitan. En este proyecto tratamos de dar a conocer al consumidor aquellos establecimientos que tiene cerca de su hogar y que probablemente desconoce, que pueden ofrecerle los mismos productos que los grandes almacenes o cadenas generalistas. Además trataremos de fomentar el envío a domicilio por parte de las empresas, dado el ámbito local en el cual trabajamos. Repasaremos la situación actual y el contexto en el que nos encontramos para realizar un análisis de requisitos para nuestro proyecto. Diseñaremos un sistema software que dé solución a las situaciones y problemáticas del modelo de negocio B2C que proponemos. Aplicando las etapas de la ingeniería del software codificaremos el sistema siguiendo la arquitectura REST y haciendo uso del patrón arquitectónico de diseño MVC, profundizando en estas dos arquitecturas.

Palabras clave—Geolocalización, ofertas, productos, empresa, consumidor, modelo, vista, controlador, entidad-relación, REST, HTTP, autentificación, API.

Abstract—It's a reality that the consumer is accustomed to search and compare consumer goods from different providers via Internet. Currently most consumers moves to a big department stores or generalist stores, that can offer the best value, to acquire those products they need. In this project we try to inform the consumer those establishments that have close his home and maybe unknown, that can offer the same products than big departments stores or generalist stores. Furthermore, we try to foment home delivery by companies, cause the local environment in which we work. We will review the current situation and the context in which we live to perform an analysis of requirements for our project. We design a software system that offer solution to situations and problematics of B2C busines model. Applying the stages of software engineering will develop the system following REST architecture and making use of the MVC architectural design pattern, delving into these two architectures.

Index Terms—Geolocation, offers, products, business, consumer, model, view, controller, entity-relationship, REST, HTTP, authentication, API.

1 Introducción

En muchas ocasiones el consumidor desconoce qué pequeños negocios tiene en su propia localidad que le pueden ofrecer los mismos productos y resolver las mismas necesidades que unos grandes almacenes o cadenas generalistas. Además la tendencia del consumidor es desplazarse a las grandes cadenas donde cree que encontrarà todo aquello que necesita.

Por eso hemos detectado la oportunidad de desarrollar un sistema basado en un modelo B2C (business to consumer, transacciones entre empresas y consumidores realizadas electrónicamente por Internet) para fomentar el consumo a nivel local ofreciendo a las PYME una herramienta para captar clientes de su zona y a su vez brindar al consumidor todas las ventajas de este modelo [1].

Desarrollaremos un sistema consistente en crear un portal web donde pequeñas PYMES puedan anunciar sus ofertas y catálogo de productos. Esto ayudará a fomentar la presencia en Internet de estos pequeños negocios sin la necesidad de que tengan que contar con una tienda virtual, algo que tendría un coste económico más elevado.

A su vez, el sistema proporcionará herramientas para las PYMES, para gestionar sus ofertas, catálogo, incidencias y pedidos, fomentando el transporte al consumidor de sus ofertas o productos, dado el ámbito local en el que se orienta. De esta manera la pequeña PYME puede aumentar su demanda ofertando transporte gratuito dentro de su localidad.

El sistema mostrará al usuario qué ofertas y productos tiene próximos a su ubicación así como si cuentan con transporte a domicilio, facilitando la tarea de adquisición del mismo. De esta manera las pequeñas PYME competirían a nivel local para que sus ofertas y productos apareciesen en las primeras posiciones.

- *E-mail de contacto: francisco.sanchezga@e-campus.uab.cat*
- *Mención realizada: Ingenieria del Software.*
- *Trabajo tutoritzado por: Oriol Ramos Terrades (CVC)*
- *Curso 2014/15*

Para realizar el sistema de manera robusta creemos conveniente seguir una arquitectura REST, de este modo podemos aprovechar todo el potencial del protocolo HTTP, que la mayoría de dispositivos contempla de forma nativa. Además nos permite separar cliente de servidor y nos ofrece la posibilidad de desarrollar en cualquier tipo de tecnología o lenguaje. Pudiendo incluso desarrollar aplicaciones especificas para sistemas operativos de terminales móviles, las cuales serán atendidas por la misma API REST. Para conseguir un desarrollo correcto y escalable vemos adecuado utilizar a su vez el patrón MVC. Que aportará muchas ventajas en la etapa de mantenimiento del sistema dado que separa la lógica, de la interfaz, dividiendo las aplicaciones en diferentes niveles. Muchos de los framework actuales utilizan estas arquitecturas por defecto.

2 Estado del arte

En la actualidad es habitual buscar información sobre productos, servicios y ofertas a través de Internet, desde la comodidad del hogar, siendo muy fácil comparar precios y características de diferentes proveedores. Es una realidad que el comercio electrónico está a la orden del día, modelos de negocio B2C están en auge. Actualmente son muchas las compañías que se dedican a ello, la más famosa Amazon, aunque podemos encontrar un sinfín de ellas, Pixmania, Mercadolibre, Segundamano, etc.

Son muchas las ventajas de este modelo de negocio, entre ellas, el consumidor encuentra una gran cantidad de información disponible a cualquier hora y tiene la posibilidad de obtener tarifas con descuentos, compras más cómodas y rápidas, precios siempre actualizados. Así como los proveedores pueden ahorrarse comisión de intermediarios y simplificar procesos.

Actualmente no hemos encontrado un portal dedicado a fomentar el comercio local, algunos proyectos pero ninguno acabado de desarrollar. Todas las compañías mencionadas tratan un público general y la mayoría cuentan con una gran logística para atender pedidos en la mayor área posible. Esto puede ser debido a que encontrar la fórmula para el éxito, limitando el público al que se orienta, requiere de un trabajo de análisis de requisitos exhaustivo para orientar la información pertinente al público adecuado [2].

A día de hoy las arquitecturas más populares para desarrollar servicios web son REST y SOAP. Ésta última cuenta con ventajas como contar con un lenguaje, plataforma y transporte independiente, mientras que REST requiere de HTTP. Funciona bien en sistemas distribuidos, utiliza estándares y cuenta con herramientas de gestión de errores, entre otros. Por contra, define XML como formato, una vez implementado el realizar algún cambio puede impactar sobre el cliente [14]. REST es más ligero, más fácil de interpretar para los humanos, no es necesario aprender un conjunto nuevo de instrucciones, sino usar las própias de HTTP, permite diferentes formatos y tiene mejor rendimiento.

3 Objetivos

Para dar solución a las situaciones mencionadas anteriormente, proponemos el desarrollo de un sistema software y nos plantemoas las siguientes alternativas:

- Alternativa 1: Desarrollo de una aplicación web accesible desde PC y dispositivos móviles desde cero totalmente personalizada siguiendo REST y MVC.
- Alternativa 2: desarrollo de una aplicación web accesible desde PC y dispositivos móviles bajo Zend Framework [12].

Debemos centrarnos en desarrollar un sistema robusto y escalable, unas herramientas de gestión para que las empresas fácilmente puedan administrar sus características y a su vez, cuenten con una interfaz de usuario clara, sencilla y funcional que permita al usuario reconocer y adquirir todo aquello que sea de su interés.

También debemos mencionar que hemos decidido aplicar la alternativa 1 por las siguientes razones: Consideramos que para lograr el objetivo principal del proyecto resulta conveniente realizar una codificación partiendo desde cero, plantearse qué tipo de arquitectura se seguirá y qué problemas podemos encontrarnos. Dado que el uso de frameworks, una vez aprendido, resuelve mucha de esta problemática, podríamos correr el riesgo de no tener claro el "por qué" de realizar la codificación de la manera en la que el framework nos indica, generando una pequeña capa de abstracción que queremos evitar.

Además el segundo motivo por el cual hemos decidido no utilizar ningún framework es que en el mundo laboral cada empresa utiliza aquel que el jefe de proyecto o la dirección consideran oportuno, por lo tanto aprendiendo las bases de REST y MVC, en un futuro, si se nos impone un framework concreto tendremos la habilidad de adaptarnos y entender de mejor manera cómo el framework concreto gestiona ciertos aspectos.

Así, a continuación, listaremos los objetivos del sistema y los catalogaremos según su prioridad (ver Tabla 1).

- Objetivo 1 Profundizar en el conocimiento, aplicación y técnicas de desarrollo del patrón Modelo Vista Controlador y arquitectura REST.
- Objetivo 2: Desarrollar herramientas para las PYME de gestión de catálogo de productos, servicios, ofertas, incidencias y pedidos.
- Sub-Objetivo 2.1: Dar a conocer al usuario aquellos establecimientos próximos a su ubicación.
- Objetivo 3: Facilitar plataforma de difusión y marketing para pequeñas y medianas empresas interesadas en mejorar su posición en el mercado, consiguiendo que por lo menos el 50% de las empresas noten mejoría en su posición.
- Obejtivo 4: Auto-sostenibilidad del sistema. Lograr que el coste de mantenimiento del sistema sea mayor o igual al beneficio obtenido en un periodo de un año.

TABLA 1
PRIORIZACIÓN DE OBJETIVOS

Prioridad:	Crítica	Media	Secundaria
Objetivo 1	X		
Objetivo 2	X		
Sub-Obj 2.1		X	
Objetivo 3			X
Objetivo 4			X

4 METODOLOGIA

En términos generales, las metologías tradicionales se basan en un ciclo de vida de desarrollo de software en cascada ya que organizan los proyectos por etapas que se ejecutan secuencialmente. Estas etapas se ejecutan una sola vez y hasta que no finaliza una etapa no se pasa a la siguiente. Aunque presentan inconvenientes como dificultades en volver a ejecutar de nuevo una etapa, puede obligar a rediseñar el proyecto entero o incluso a descartar parte del progreso logrado. El usuario no ve el producto hasta el final, y que si el ámbito del proyecto es muy cambiante pueden variar las necesidades del proyecto que se definieron inicialmente.

Por otra parte las metodologías ágiles se basan en un ciclo de vida iterativo e incremental, se hacen entregas parciales del producto para ir validando con el cliente que el producto cumple con los requisitos, se pueden solapar etapas, y se cambia la documentación por la interaccion con el usuario. No obstante existe el riesgo de entrar en un ciclo en el que solo se presentan prototipos, además existe el riesgo de falta de documentación de los diseños, problemas derivados de la comunicación oral y falta de reusabilidad a causa de la falta de documentación. Asi como limitaciones en el tamaño de los proyectos.

Dado el calendario propuesto, que no contamos con un equipo, que se nos solicita una documentación específica y que creemos que la definición y calidad en lugar de la velocidad, es la clave para el éxito, hemos decidido seguir esta clásica metodología Waterfall donde se han aplicado los bloques de la Ingenieria del Software: análisis de requisitos, diseño, codificación y test.

De todas maneras y en previsión de las debilidades de Waterfall, en la etapa inicial hemos dedicado un amplio tiempo al bloque de análisis de requisitos. Aunque posteriormente decidimos adaptarnos al Waterfall iterativo, dada la amplitud y cantidad de funcionalidades del proyecto (ver Figura 1), de tal manera que consideramos el progreso realizado como la primera iteración del proyecto [3].

El desarrollo se ha llevado a cabo de forma que ha habido actividades individuales que han requerido de la generación de artefactos tales como documentos, hojas de cálculo, diagramas, imágenes, etc., que se han almacenado en una carpeta en Google Drive, donde todo artefacto generado incluye información de control y versiones.

El desarrollo se ha llevado a cabo siguiendo en todo momento la LO 15/1999 de Protección de Datos Personales.

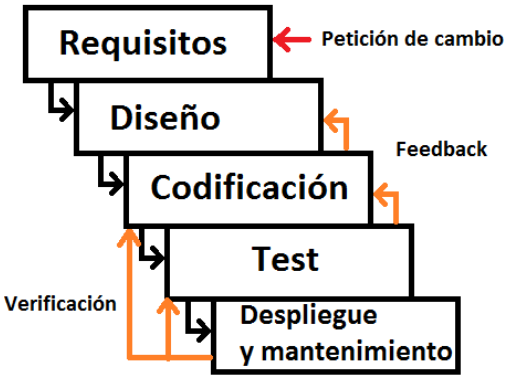


Fig. 1. Diagrama Waterfall adaptado con iteraciones.

5 PLANIFICACIÓN

Una vez definidos los objetivos, la alternativa escogida y decidido qué metodología seguiremos, realizamos una planificación de las etapas, actividades y tareas del proyecto que podemos encontrar en el apéndice A1. Podemos encontrar en la Figura 2 el calendario del proyecto, el cual es recomendable ver para tener una idea general de la distribución del tiempo.

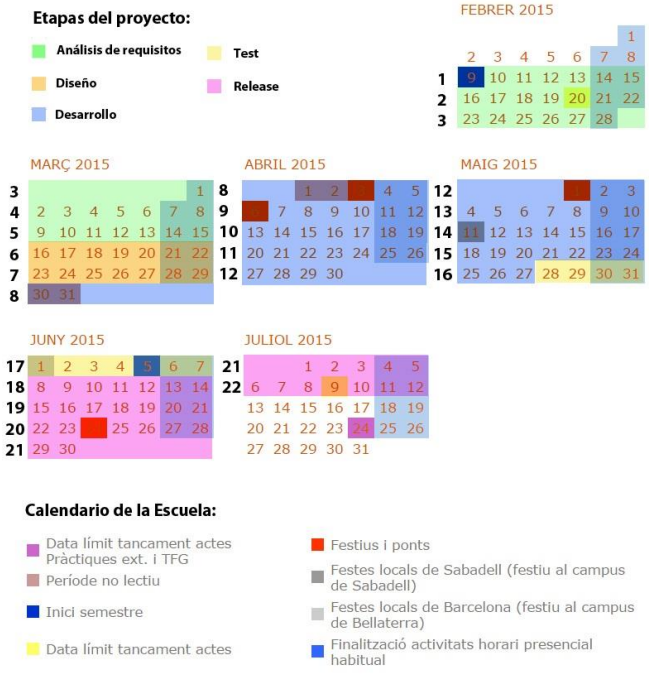


Fig. 2. Calendario.

Como componente crucial cabe destacar las milestone o hitos que hemos hecho coincidir con los hitos del TFG y que detallamos a continuación:

- Milestone 1: Actividades relacionadas con la etapa de requisitos del software, presentación del tema del proyecto, captura de objetivos y requisitos, documentación y elaboración del informe inicial. 13/Marzo/2015.
- Milestone 2: Actividades relacionadas con la etapa de deiseño y principio de la etapa de codificación, diseño de la base de datos, diseño del árbol de pantallas, diseño del backend, codifiación del script de la base de datos, de la pantalla principal, esqueleto del sistema y preparar documentación del primer informe de seguimiento. 26/Abril/2015.
- Milestone 3: Actividades relacionadas con la etapa de codificación y test. Terminar todos los modulos y vistas del sistema, realizar exploratory testing y preparar el informe de progreso 2. 29/Mayo/2015.
- Milestone 5: Artículo, confección del artículo de 10 páginas de la explicación del trabajo, agradecimientos y bibliografía. 19/Junio/2015.
- Milestone 4: Dosier del TFG, recopilación de toda la documentación para conformar el dosier final del TFG, confección de la lista de cambios. 26/Junio/2015
- Milestone 6: Defensa pública, presentación del trabajo realizado delante del tribunal. 3/Julio/2015.

Por último nos vemos en la obligación de hacer mención a la lista de riesgos y su catalogación (que podemos encontrar en la Tabla 3):

- R1: Falta de formación con las herramientas.
- R2: Menos reutilización de la prevista.
- R3: Cambios constantes en los requisitos.
- R4: Desarrollo incorrecto de las funciones del software.
- R5: Desarrollo incorrecto de las interfaces de usuario.
- R6: Diseño de dudosa escalabilidad.
- R7: Documentación escasa.
- R8: Omisión de la etapa de test.
- R9: Retrasos para alcanzar las milestone.
- R10: No encontrar demanda del software.
- R11: Falta de stakeholders reales.

TABLA 3
CATALOGACIÓN DE RIESGOS

Probabilidad Severidad	Muy Probable	Probable	Poco Probable
Crítica	R1	R8	R4, R5, R7
Media	R3, R9	R11, R6	
Marginal	R10	R2	

Debemos destacar que durante el transcurso del proyecto se nos presentaron los riesgos *R6: Diseño de dudosa escalabilidad* con el diseño de la base de datos por lo que hemos tenido que rediseñarla y volver a generarla, *R1: Falta de formación con las herramientas* nos hemos encontrado con que hemos tenido que invertir más tiempo del previsto en la formación sobre MVC y REST. Consecuentemente hemos sufrido el riesgo *R9: Retrasos para alcanzar las milestone,* que como ya lo tuvimos en cuenta dado que las milestone fueron planificadas con una semana de margen y contamos con un plan de contingencia definido, el impacto de las consecuencias resultó minimizado.
Además nos dimos cuenta de que cometimos un error en la estimación de las tareas en función a los recursos del proyecto. Aun así hemos alcanzado el Objetivo 1 y el Sub-Objetivo 2.1 de manera completa y el Objetivo 2 no de manera completa. Descartando completamente, en esta iteración, los Objetivos 3 y 4.

6 REQUISITOS

Para realizar el análisis de requisitos una vez estudiado el contexto y la situación actual nos hemos apoyado en una serie de encuestas realizadas a empresas y usuarios que podemos encontrar en el anexo A2. La encuesta para empresas se centra en con qué cuentan actualmente, si pagina web, tienda virtual, tipo de base de datos, además de en sus intereses referentes a la presencia en internet de su negocio. Y la encuesta para usuarios se centra sobre sus hábitos de consumo en Internet.

A continuación mencionaremos algunos requisitos del sistema, debiendo diferenciar dos tipos de usuarios, el usuario consumidor y el usuario empresa, por lo tato los requisitos indexados por 1.X pertenecen al usuario consumidor y los requisitos indexados por 2.X pertenecen al usuario gestor de empresas.

A continuación destacaremos los requisitos principales pudiendo encontrar el total de ellos en el anexo A3.

6.1 Funcionales

- RF 1.1: Tan pronto como el usuario acceda a la aplicación, el sistema tiene que mostrar al usuario un listado de productos, servicios y ofertas teniendo en cuenta su proximidad. Prioridad: Alta. Riesgo: Alto
- RF 1.2: El sistema tiene que mostrar los productos, servicios y ofertas situados en un mapa. Prioridad: Alta. Riesgo: Medio.
- RF 1.4: El sistema tiene que proveer al usuario con la habilidad de buscar productos, servicios y ofertas por nombre. Prioridad: Media. Riesgo: Alto.
- RF 1.13: El sistema tiene que proveer al usuario con la habilidad de realizar pedidos o reservas. Prioridad: Alta. Riesgo: Alto.
- RF 2.1: El sistema tiene que proveer al usuario empresa con la habilidad de introducir información relativa a su local/comercio como nombre, descripción, horarios, localización. Prioridad: Alta. Riesgo: Alto.
- RF 2.2: El sistema tiene que proveer al usuario empresa con la habilidad de gestionar su catálogo de productos servicios y ofertas. Prioridad: Alta. Riesgo: Alto.

6.2 No funcionales

- RNF1: La web y base de datos deben estar operativas en cualquier momento, permitiendo al usuario poder interaccionar con el sistema sin esperar más de 10 segundos para cualquier operación (siempre y cuando la

conexión de red y el proveedor de Internet lo permitan). Prioridad Alta. Riesgo: Alto.
- RNF2: Todos los datos personales sensibles, así como las contraseñas, deberán guardarse cifrados en la base de datos y nunca deberán enviarse en claro a través de la red. Prioridad: Alta. Riesgo: Alto.

7 Diseño del sistema

Como ya hemos especificado el sistema contará con una arquitectura REST y seguirá el patrón MVC para backend. Hemos realizado el diseño entidad-relación de la base de datos y una vez generada hemos extraido el diseño de tablas. Por parte del frontend hemos realizado prototipos en papel de la interfaz de usuario y los hemos modelizado con la herramienta web Cacoo [13] ya que ofrece justo las herramientas necesarias para el modelado y nos ha parecido mejor que realizarlo mediante editores de imagen.

7.1 REST y MVC

Representational State Transfer (REST) , es una arquitectura habitualmente utilizada en el desarrollo de servicios web que hace total uso del protocolo o estándar HTTP. Es una manera simple de organizar interacciones entre sistemas independientes, su uso esta en auge (por ejemplo Twitter API) ya que permite interactuar con clientes de diferentes plataformas.

Además también impone que nunca debemos hacer uso de variables de sesión, la arquitectura es stateless, lo que significa que toda la información necesaria para gestionar la petición debe estar autocontenida en la misma. Y esto conlleva ciertas dificultades frente a un desarrollo web convencional basado en sesiones, como no poder mantener los datos del usuario una vez identificado, sus pedidos, o mantener cualquier tipo de información en la variable de sesión. Aun así resulta mucho más simple, escalable e igual de potente que alternativas como SOAP.

Esta arquitectura orientada a recursos se basa en varios niveles:

- Uso correcto de las URI.
- Uso correcto de HTTP.
- Implementar Hypermedia.

Las URL (Uniform Resource Locator) son un tipo de URI (Uniform Resource Identifier) que nos permiten identificar de manera única un recurso.

Siguiendo esta arquitectura debemos hacer un uso correcto de HTTP, es decir, debemos hacer un uso correcto tanto de los verbos del estándar (GET, POST, PUT, DELETE) como de las respuestas que proporcionamos. Nuestro sistema realizará una acción u otra dependiendo del verbo que se utilice en la petición. Y en consecuencia deberemos generar respuestas con el código de estado adecuado (200 OK, 307 Redirección temporal, 401 No autorizado, 404 No encontrado, etc.). Por último debemos hacer uso de hipermedia, en cada petición podemos adjuntar contenido adicional ya sea mediante XML o JSON.

El patrón MVC es un patrón de arquitectura de software que se encarga de separar la lógica de la interfaz. MVC divide las aplicaciones en tres niveles:

- Modelo: Se encarga de acceder, modificar o eliminar los datos permanentes, es decir es el encargado de realizar las consultas a la base de datos.
- Vista: Se encarga de mostrar y formatear de manera gráfica la información al usuario. Es la interfaz del sistema.
- Controlador: Es el intermediario entre la vista y el modelo. Solicita datos al modelo y los entrega a la vista. Es realmente donde se encuentra la lógica de cada acción que realiza el usuario.

Combinando REST y MVC podemos conseguir un sistema de grand calidad, robustez y escalabilidad. Aplicamos MVC para cada recurso que REST identifica por URL. De manera que contamos con un controlador frontal que aplica la arquitectura REST, gestiona la petición y la enruta al recurso adecuado el cual aplica MVC. En la Figura 2 podemos ver el diagrama del sistema [4][5][6].

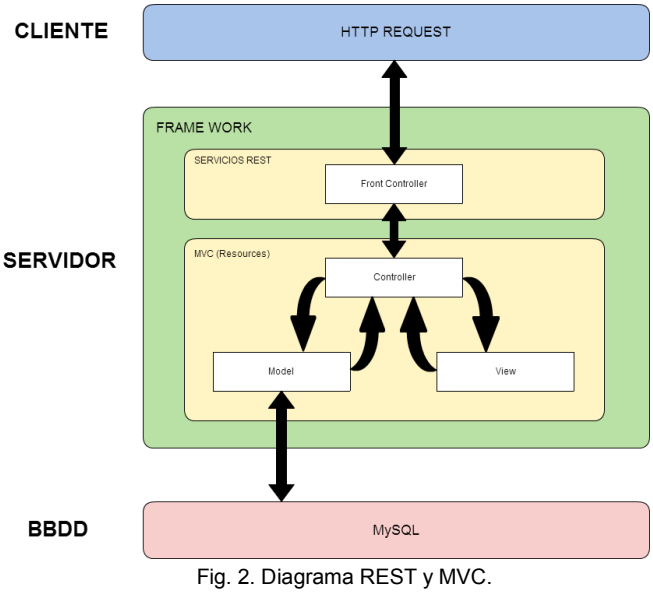


Fig. 2. Diagrama REST y MVC.

El cliente realiza una petición que es atendida por el controlador frontal (REST) que enruta la petición al recurso específico (controlador que aplica MVC) y éste devuelve la respuesta. Por lo tanto contamos con una colección de recursos que serán los servicios que ofrecerá nuestro sistema. Toda petición debe ser gestionada por el controlador frontal, incluso las llamadas AJAX. La respuesta del sistema se comporta como una API.

7.2 Base de datos

La base de datos la hemos diseñado de la siguiente manera, los productos son genéricos para cualquier empresa que disponga de ellos, así cuando una empresa ha dado de alta un producto en el sistema y otra empresa, que también vende el mismo, quiere darlo de alta ya lo puede encontrar en el sistema en lugar de dar de alta dos veces el mismo producto, consiguiendo que evite introducir

excesiva información y agilizando el trámite. Solo tendría que detallar los atributos própios como precio, imágenes, descripción y ofertas relacionadas. De esta manera evitamos redundancia en la base de datos (agregación de la figura 3). Entendemos "productos" como productos o servicios dado que ambos cuentan con los mismos atributos y no necesitamos tener dos entidades diferentes.

A su vez, el usuario puede realizar pedidos o reservas, así como ser el administrador de N empresas. Cada empresa puede disponer de N catálogos de productos y/o servicios. Y por último la puntuación que asignes el usuario a los productos se realizara de manera independiente a la empresa de la cual lo haya adquirido, ya que el sistema de puntación será la base de información del sistema recomendador.

En el diagrama de la Figura 3 hemos omitido tanto las llaves primárias como los atributos por claridad

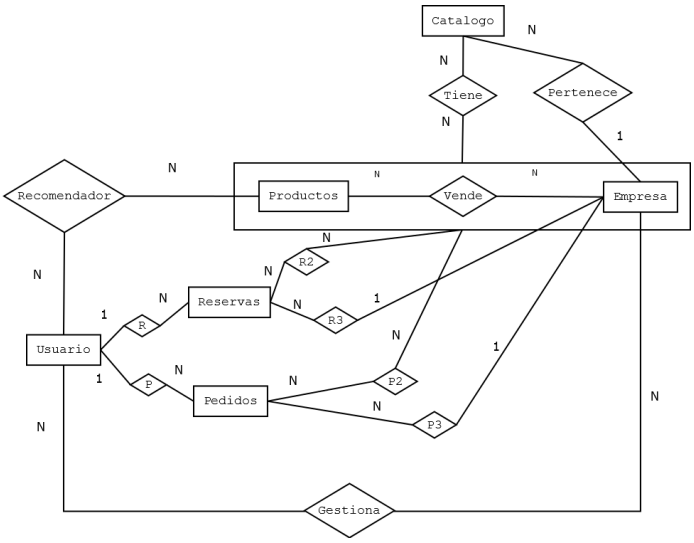


Fig. 3. Diagrama entidad-relación de la BBDD.

7.3 Wireframe

Para el diseño de la interfaz de usuario realizamos primero un prototipo en papel el cual validamos con algunos usuarios.

Una vez hecho esto, lo modelamos mediante la herramienta Cacoo. Esta herramienta web gratuita permite modelar de manera gráfica e intuitiva diferentes tipos de diagramas y esquemas.

A cada pantalla del wireframe le asociamos los requisitos funcionales que debe satisfacer.

Hemos realizado el diseño de la barra de navegación, pantalla principal, pantalla de detalle del producto, pantalla de búsqueda, pantalla de perfil de usuario (tanto usuario convencional como empresa), pantalla de añadir producto y pantalla de catálogo.

A continuación detallamos en una tabla los requisitos que debe atender cada pantalla Tabla 4.

TABLA 4
PANTALLA - REQUISITOS

Pantalla	Requisitos
Principal	RF1.1, RF1.2, RF1.4, RF1.8, RF1.9, RF1.12
Detalle de producto	RF1.5, RF1.6, RF1.7, RF1.13
Búsqueda	RF1.3, RF1.4
Perfil de usuario	RF1.10, RF2.1, RF2.5
Añadir producto	RF2.6
Catálogo	RF1.6, RF2.6

Por último y por cuestiones de formato solo mostraremos el wireframe de la pantalla principal, Figura 4.

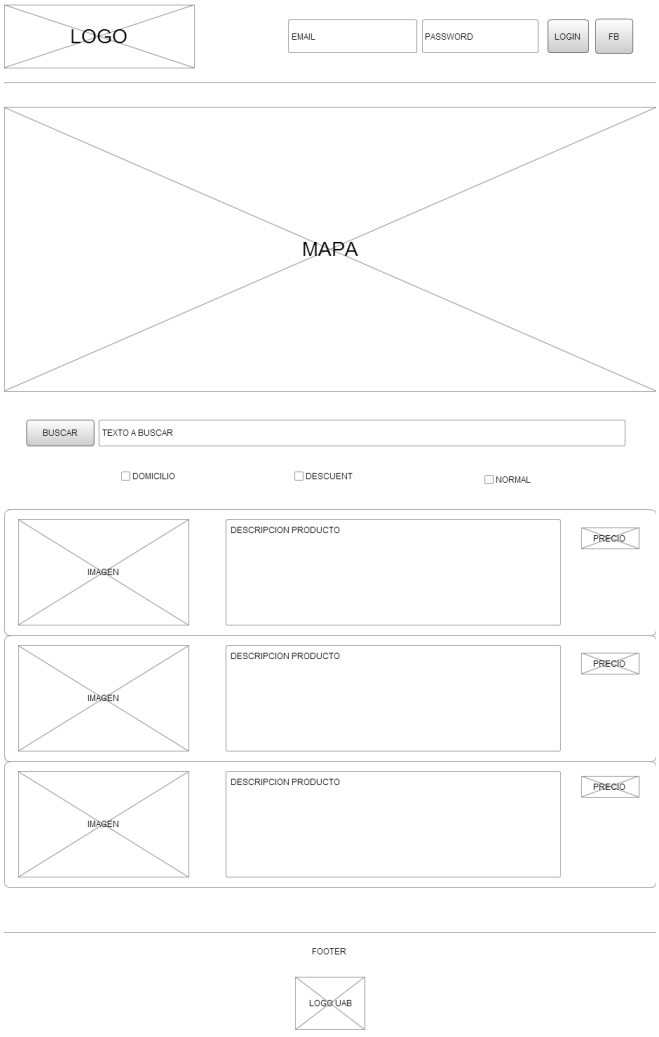


Fig. 4. Pantalla principal.

Cabe destacar que la barra de navegación sufrió un rediseño dadas las restricciones del sistema de autentificación de la arquitectura REST de las cuales hablaremos en la siguiente sección. En lugar de encontrarnos con un formulario donde introducir el identificador y password encontraremos dos botones, uno para identificarse como usuario y otro para hacerlo como empresa.

8 CODIFICACIÓN

Antes de entrar a detallar aspectos de la codificación debemos mencionar que hemos utilizado el entorno de trabajo XAMPP 5.6.3 para Windows, Apache/2.4.10, PHP 5.6.3, MySQL 5.6.21, phpMyAdmin 4.2.11 y todas las pruebas se han realizado en Google Chrome.

Lo primero que hemos realizado ha sido la base de datos y una vez realizado el diseño explicado en la sección 6.2, la hemos creado bajo phpMyadmin. Una vez hecho, hemos realizado algunas pruebas para comprobar que no existe ningún error en las relaciones.

De acuerdo con el Modelo Vista Controlador definimos la estructura de directorios como podemos ver en la Figura 5.

```
/TFG
    /config
    /controllers
    /models
    /views
        /bootstrap
        /img
        /navbar
    .htaccess
    Index.php
```

Fig. 5. Estructura de directorios.

Siendo Index.php el controlador frontal encargado de aplicar REST.

Los formularios se han realizado de manera dinámica con Javascript. Y para el estilo de las vistas, a parte de utilizar CSS, se ha hecho uso de Bootstrap que es un pequeño front-end framework gratuito que incluye HTML, CSS, basado en el diseño de plantillas y con soporte para plugins JavaScript. De manera que las vistas las hemos realizado de manera *responsive*, es decir, que se adaptan al tamaño de pantalla.

Así mismo hemos utilizado HTML y la API de Google Maps para la creación del mapa de la página principal el que consta de iconos personalizados, creador de rutas y calculador de distancia y tiempo en coche.

8.1 Controlador Frontal

La mayor parte del tiempo la hemos invertido en el controlador frontal, dado que cualquier cambio en él afecta al resto de la aplicación. En varias ocasiones nos hemos visto en la situación de añadir recursos sin contar con un controlador frontal suficientemente robusto y tener que descartar el progreso, ya que estábamos desarrollando de manera incorrecta puesto que el controlador frontal no respondía como era debido. En ocasiones, el controlador frontal, no invocaba al controlador correcto, invocaba a más de un controlador, o incluso entraba en bucle infinito bajo ciertas circunstancias en el parseo de las URIs cuando un recurso generaba una nueva petición.

Recordamos que cualquier petición o acción que se realiza la gestiona y enruta dicho controlador. Por lo tanto para redireccionar todas las peticiones al controlador frontal debemos haber creado el archivo .htacces, que es un archivo que permite definir diferentes directivas de configuración para cada directorio, como podemos ver en la Figura 6.

```
RewriteEngine On
RewriteBase /tfg

RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule ^(.*)$ index.php/$1 [L]
```

Fig. 6. Archivo .htaccess.

Para el sistema de autentificación hemos tenido que hacer que sea el propio navegador del cliente el que solicite la información necesaria para generar las cabeceras adecuadas (WWW-Authenticate). Para ello PHP cuenta con el método *header* que hemos utilizado de la siguiente manera:

```
header('WWW-Authenticate: Digest
realm="'.$realm.'",qop="auth",nonce="'.uniqid().'",opaque=
"'.md5($realm).'"');
```

Donde *$realm* identifica a qué páginas tendrá acceso el usuario y *nonce* es un número generado de manera única para cada petición. De esto hablaremos en mayor profundidad en la siguiente sección 8.2.

El controlador frontal lo primero que hace es incluir los archivos que se encuentran en el directorio config, que son la conexión a la base de datos y una clase genérica que controla si la petición esta autentificada o no. En función de ello se mostrará una barra de navegación u otra, además concederá acceso a unas áreas del sistema u otras. A continuación incluímos de manera automática los ficheros de los directorios controllers y modelos del recurso apropiado. De esta manera no tenemos que incluirlos en cada fichero. Este controlador cuenta con una clase *Request* que se encarga de analizar la petición HTTP que realiza el cliente, detecta el verbo de la petición, la URI a la que se pretende acceder y, en caso de haberlos, parsea los parámetros en JSON o XML. Por último analiza la URI de la petición y, si el recurso al que tratamos de acceder existe, enruta la petición al controlador del recurso (controlador MVC).

Los recursos, que aplican MVC, están desarrollados mediante programación orientada a objetos. Los controladores y modelos son clases mientras que las vistas se desarrollan bajo HTML, CSS y Javascript.

Por lo tanto, cuando el cliente realiza una petición como *GET dominio/tfg*, el controlador frontal detecta que se pretende acceder al recurso tfg (página principal) y que la acción es GET.

Todos los controladores de los recursos deben tener los métodos públicos que deban gestionar de la siguiente manera *verboAction*. A modo de ejemplo podemos ver las líneas 10 y 18 de la Figura 7, ya que el controlador frontal invocará esta acción.

Para entender mejor la estructura de los controladores mostramos, a modo de ejemplo, parte del controlador de la página principal:

```
0: class TfgController {
1:   public $model;
2:
3:   public function __construct()
4:   {
5:      $this->model = new TfgModel();
6:   }
7:
8:   public function getAction()
9:   {
10:    $lista = $this->model->getListaprod();
11:    $lista = $this->show($lista);
12:    $ret = Log::navbar();
13:    $navbar = $ret[0];
14:    $user = $ret[1];
15:    include 'views/tfgView.php';
16:    }
17:
18:    public function postAction($parameters)
19:    {
20:    $c1 = parameters["check1"];
21:    $c2 = parameters["check2"];
22:    $c3 = parameters["check3"];
23:    $array = array($c1,$c2,$c3);
24:    $lista = $this->checkBox($array);
25:    $lista = $this->show($lista);
26:    echo $lista;
27:    }
```

Fig. 7. Ejemplo parcial de controlador.

En este caso, una petición GET a este recurso, mostrará la página principal mientras que reservamos el verbo POST para atender las peticiones AJAX de la página principal, para filtrar los resultados en función del tipo.

En este mismo ejemplo podemos ver cómo funciona el patrón MVC. Cuando el controlador necesita acceder a datos de la BBDD invoca al modelo que realiza la consulta, línea 10 de la Figura 7, éste devuelve los datos, el controlador opera con ellos, línea 11 de la Figura 7 y llama a la vista, línea 15 de la Figura 7.

Nos ha supuesto dificultades adaptarnos a la arquitectura. Cada petición debe ser atendida por el controlador frontal teniendo en cuenta que estamos trabajando con peticiones independientes y que no podemos guardar ningún tipo de estado. Pero de esta manera conseguimos que el sistema sea más liviano, además de poder atender cualquier acción en una misma petición. Cuando el cliente realiza una petición GET con parámetros y, posteriormente, utilizamos una carga dinámica AJAX (JQuery) que realiza una nueva petición para cargar contenido en la página, al tener que mostrar en una misma vista el resultado de dos peticiones diferentes, hemos tenido que utilizar Javascript (banda del cliente) para recordar la primera petición. Esto nos ha sucedido en la página de búsqueda, por ejemplo, que podemos ver en la Figura 8 [7][8][9].

Fig. 8. Dos peticiones en la misma vista.

8.2 Autentificación

Para la autentificación del usuario hemos tenido que utilizar autentificación mediante HTTP, que cuenta con dos sistemas de autentificación, básico y digest. Por cuestiones de seguridad y en cumplimiento con el requisito no funcional RFN2 hemos optado por HTTP Digest. Este método realiza los siguientes pasos:

- El cliente realiza una petición al servidor.
- El servidor responde con un código especial (nonce) único en cada petición, el *realm* (reino al que concede acceso, recursos disponibles para esa autentificación) y pregunta al cliente por sus credenciales.
- El cliente responde con el código especial y el nombre de usuario, contraseña y realm encriptados.
- El servidor responde con la información solicitada si coinciden los datos o con error 401 en caso contrario.

El controlador de autentificación realiza una consulta para saber qué usuarios están registrados en el sistema, especifica el *realm* (usuario o empresa) y, si la petición no contiene la cabecera de autentificación, rellena la cabecera de la manera anteriormente explicada (sección 8.1) para solicitar autentificación al usuario y, en caso del usuario

cancelar la autentificación, lo redirige a la página principal. Si no, analiza la variable PHP_AUTH_DIGEST que contiene todos los elementos necesarios para la autentificación y si coincide con un usuario existente en la base de datos, genera una respuesta valida encriptado mediante md5 el usuario, realm, contraseña, verbo de la petición, URI y el *nonce*. Una vez el usuario esta autentificado, en cada petición se sigue controlando que las credenciales sean válidas, ya que son únicas en cada petición.

Por otra parte el estándar de HTTP Authentication (rfc2617) no especifica el método de logout. Y lo que hemos hecho ha sido actualizar el header a no autorizado cuando el usuario decide desconectarse del sitio [10][11].

9 RESULTADOS

Pese a las dificultades mencionadas en la sección 5 por la aparición de algunos riesgos, no hemos podido implementar todos los requisitos funcionales que hubiéramos deseado. Aunque los objetivos principales del TFG se han alcanzado.

Hemos realizado por completo la etapa de análisis de requisitos habiendo realizado dos modelos de encuestas de los cuales extraer nuestros requisitos, incluso podemos contar con *stackholders* reales para la actividad de validación de la etapa de test. También hemos completado la etapa de diseño del sistema, contando con un diseño de base de datos escalable. Contamos con el diseño del árbol de pantallas y, por parte del backend, hemos comprendido cómo utilizar estas dos arquitecturas REST y MVC de manera correcta. Además contamos con un esqueleto o núcleo del sistema que aplica REST y MVC de manera correcta y robusta. Aunque tenemos un error detectado en la etapa de test, en el cual el sistema no se comporta como se espera cuando en la URI se llama dos veces al mismo recurso, por ejemplo: *GET dominio/tfg/perfil/perfil*. Hemos detectado el error y pensado su solución.

Como comentamos en la sección 8.1 y 8.2, cuando el usuario utiliza el botón de autentificación (Particulares o Empresas), se establece el *realm* al que tiene acceso, determinando los recursos a los que podrá acceder. Una vez autentificado, la barra de navegación se establece en función del tipo de usuario (*realm*), mostrando unas opciones u otras. El usuario estándar, a diferencia del usuario empresa, no cuenta con acceso a los recursos de gestión.

Seguidamente mostraremos algunas capturas de pantalla de la aplicación, la autentificación vía navegador podemos verla en la Figura 9 (PRIMERAMANO Particulares es el *realm*), el navegador abre un cuadro de diálogo preguntándonos por nuestras credenciales.

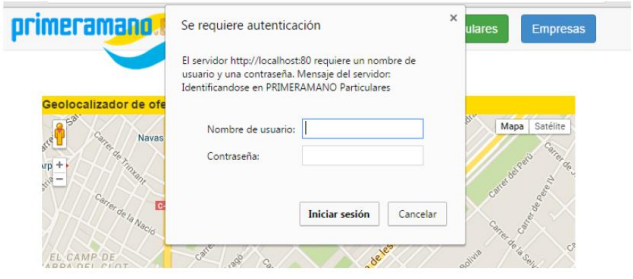


Fig. 9. Autentificación.

La pantalla principal sin identificación por parte del usuario tal y como se diseño en el Wireframe (sección 7.3), incluyendo el cambio ya mencionado de la barra de navegación, podemos verla en la Figura 10.

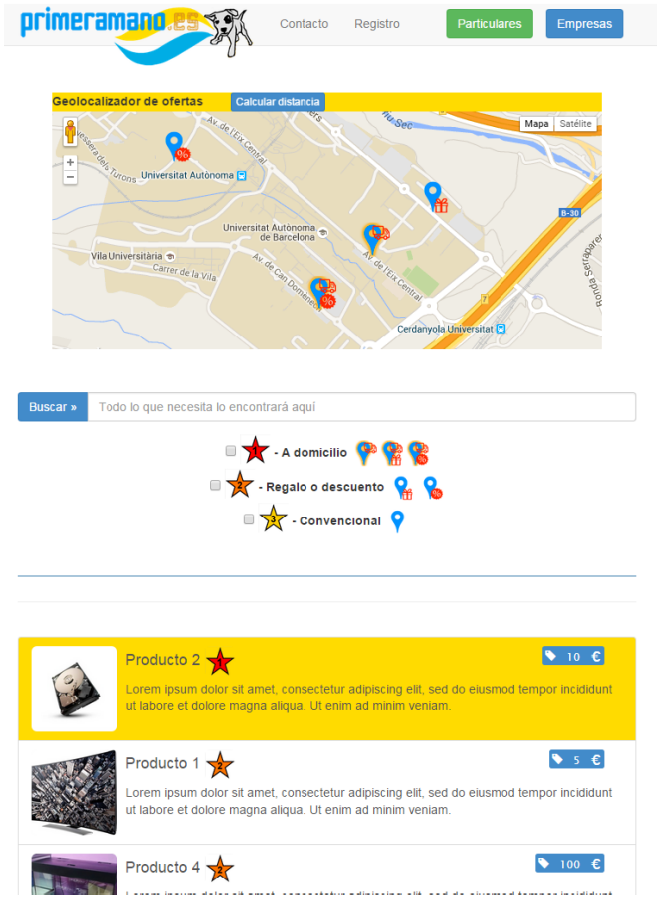


Fig. 10. Pantalla principal.

Por último, por cuestiones de formato, mostramos en la Figura 11 la pantalla de perfil del usuario autentificado. Donde cuenta con la habilidad de modificar alguno de sus datos mediante un formulario dinámico (a diferencia del formulario validado de registro). Además de poder ver y seguir el estado de sus pedidos en caso de tenerlos.

Fig. 11. Pantalla de perfil usuario.

10 TRABAJO FUTURO

En siguientes iteraciones del proyecto se trataría de ir añadiendo recursos al sistema para acabar de completar todos los requisitos funcionales. Corregir los errores detectados mediante el *exploratory testing*. Alcanzar el nivel 3: Hypermedia, ya que nos ha sido más sencillo devolver vistas directamente, aunque es subsanable.

Realizar una etapa de validación con usuarios reales. Dependiendo del feedback recibido y del comportamiento del sistema realizar una nueva iteración para implementar los posibles cambios o rediseños. Y por último realizar el despliegue.

11 CONCLUSIÓN

Nos hemos dado cuenta que implementar un framework desde cero es una carga de faena que requiere de mayor cantidad de recursos humanos o de tiempo. No obstante hemos aprendido muchísimos detalles sobre la arquitectura REST y MVC, aunque no hayamos podido completar todos los objetivos, tenemos la certeza de que en cada paso que hemos dado se han seguido las etapas de la Ingeniería del Software y se ha llevado a cabo de manera correcta sin recurrir a soluciones, que aunque funcionales, de menor calidad, escalabilidad y robustez.

Hemos podido comprobar la diferencia entre realizar una aplicación web de manera convencional, con los conocimientos que teníamos hasta ahora, basada en sesiones, AJAX, y sin usar la programación orientada a objetos. A realizarla bajo una arquitectura como es REST y aplicando MVC, con lo que se puede llegar a desarrollar servicios web de gran calidad.

Para futuros proyectos web utilizaremos un framework dada la cantidad de funcionalidades que resuelve, la documentación disponible y el asesoramiento de la comunidad. Pese a ello creemos que nos ha sido un gran ejercicio académico ya que nunca habíamos utilizado estas arquitecturas y hemos podido comprobar el potencial que pueden llegar a ofrecer. También hemos comprendido el funcionamiento de ambas y aun utilizando un framework en futuros proyectos, volveríamos a apostar por REST y MVC. Creemos que gracias a este proyecto podremos comprender mucho mejor el funcionamiento de los diferentes frameworks actuales.

Por otra parte, nos ha provocado muchas dificultades adecuarnos al principio *stateless* y utilizar las características de HTTP. Pero una vez logrado, comprendemos que es una muy buena práctica que cualquier desarrollador web debería conocer en cierta profundidad.

AGRADECIMIENTOS

Agradecer a las empresas que nos han facilitado sus datos de contacto en las encuestas y que han mostrado interés en seguir la evolución del proyecto, pese a no haber alcanzado el grado de madurez deseado para ponernos en contacto con ellas: Musicman, Qmqxul, Stefany-disfraces, Espaimoscota, Option1, Dimesa, Moniberic, Limit Surf, Llibreria L'Illa, Zulex.

Cabe remarcar que, tanto el logo como la elección de colores para el estilo de página ha sido reaprovechada de una web *mock* que realicé para la presentación del Trabajo fin de Master en Organización Industrial para la UPC de Nabil Ingranchen López, quien abordó el concepto desde un punto de vista más comercial y a nivel empresarial.

Por último agradecir el consejo e interés desinteresado por parte del tutor Oriol Ramos Terrades.

BIBLIOGRAFIA

[1] Joan Lozoya – Artículo, Clases de comercio electrónico. http://suite101.net/article/clases-de-comercio-electronico-b2b-b2c-b2a-b2e-c2c-c2g-b2g-a26589#.VYXmz_ntlBd

[2] Fernando Pérez Nava – Presentación, Comercio electrónico. http://fdoperez.webs.ull.es/doc/Modelos%20de%20Comercio%20Electr%F3nico.pdf

[3] SEGUE TECHNOLOGIES – Blog, Waterfall vs. Agile: Which is the Right Development Methodology for Your Project?, 5 de Julio de 2013 http://www.seguetech.com/blog/2013/07/05/waterfall-vs-agile-right-development-methodology

[4] Victor Robles – Blog, ¿Qué es el patrón MVC?, 1 de Febrero 2014. http://victorroblesweb.es/2013/11/18/tutorial-mvc-en-php-nativo/

[5] PHP-HTML – Tutorial, Model View Controller (MVC) in PHP, 12 de Agosto 2009. http://php-html.net/tutorials/model-view-controller-in-php/

[6] Ludovico Fischer – Blog, A Beginner's Guide to HTTP and REST, 9 de Junio 2013. http://code.tutsplus.com/tutorials/a-beginners-guide-to-http-and-rest--net-16340

[7] Eugenia Bahit – E-Book, POO y MVC en PHP, El paradigma de la Programación Orientada a Objetos en PHP y el patrón de arquitectura de Software MVC, Primera edición no revisada, 15 de Junio 2011. http://eugeniabahit.blogspot.com.es/2011/07/poo-y-mvc-en-php.html

[8] Lorna Jane Mitchell – Blog, Building a RESTful PHP Server, 19 de Junio 2012. http://www.lornajane.net/posts/2012/building-a-restful-php-server-understanding-the-request

[9] Stackoverflow – Foro, REST API Login Pattern, 26 de Diciembre 2012. http://stackoverflow.com/questions/13916620/rest-api-login-pattern

[10] Joshua Thijssen – Blog, The RESTful CookBook. http://restcookbook.com/Basics/loggingin/

[11] Network Working Group – Estandard RFC2617, HTTP Authentication: Basic and Digest Access Authentication, Junio 1999. http://tools.ietf.org/html/rfc2617/

[12] Zend Framework – Web, Zend Framework Home. http://framework.zend.com/

[13] Cacoo – Web, Cacoo Home. https://cacoo.com

[14] The World Wide Web Consortium – Nota, Simple Object Access Protocol (SOAP) 1.1. 08 de Mayo 2000. http://www.w3.org/TR/2000/NOTE-SOAP-20000508/

APÈNDICE

A1. PLANIFICACIÓN

TABLA 2
WORK BREAK STRUCTURE

Análisis de requisitos	**Duración: 5 semanas**
Actividad	**Tareas**
Presentación del tema del proyecto	• Definición formal del proyecto. • Reunión con el tutor, explicación del tema del proyecto, herramientas, ideas.
Documentación	• Gestión de Google Drive. • Gestión de la documentación del proyecto. • Análisis de la situación actual y contexto. • Confección del informe inicial.
Captura de objetivos y requisitos	• Confección de encuestas (usuarios y encuestas). • Realización de encuestas a empresas a pie de calle. • Realización de encuestas a empresas y usuarios de manera online. • Análisis y documentación de resultados.
Milestone 1	• Entrega del informe inicial: Introducción, situación actual, objetivos, requisitos y planificación. • Reunión con el tutor a modo de seguimiento.
Diseño	**Duración: 2 semanas**
Actividad	**Tarea**
Diseño de la base de datos	• Diseño ER y de tablas de la base de datos.
Diseño del árbol de pantallas	• Diseño sobre papel de las pantallas de la interfaz (vistas). • Diseño con herramienta de modelado de las pantallas.
Diseño del backend	• Búsqueda de información del patrón MVC. • Búsqueda de información sobre arquitectura REST. • Detección de recursos principales.
Documentación	• Confección del informe de seguimiento 1.
Desarrollo	**Duración: 8.5 semanas**
Actividad	**Tareas**
Script inicial de la base de datos	• Desarrollar el script inicial de creación de la base de datos MySQL. • Desarrollar script de test.
Desarrollo de la vista principal	• Desarrollar la vista principal. HTML5+CSS3. • Desarrollar el script inicial JavaScript del mapa de Google (Google Maps API).
Desarrollo de los módulos prioritarios	• Desarrollar registro en el sistema, login y login con Facebook. • Desarrollar recurso para mostrar los productos, servicios y ofertas destacados de la pantalla principal en función de la proximidad.
Milestone 2	• Entrega del informe de seguimiento 1. • Reunión con el tutor a modo de seguimiento.
Desarrollo de todas las vistas del sistema	• Desarrollar todas las vistas diseñadas del sistema.
Desarrollo de todos los recursos del sistema	• Desarrollar todos los recursos del sistema por orden de prioridad (requisitos funcionales ordenados por prioridad).
Documentación	• Confección del informe de seguimiento 2. Actualización si es necesario de los documentos anteriores. • Comenzar la confección del artículo.
Milestone 3	• Entrega del informe de seguimiento 2. • Reunión con el tutor a modo de seguimiento.
Test	**Duración: 1.5 semanas**
Actividad	**Tareas**
Exploratory Testing	• Realizar test exploratorio del sistema en busca de errores y fallos. • Corrección de los errores detectados.
Test de validación	• Presentar el sistema a usuarios y empresas.
Documentación	• Confeccionar documento de test, lista de errores y opinión de usuarios y empresas. • Actualización de documentos anteriores en caso de ser necesario.
Release	**Duración: 5 semanas**
Actividad	**Tareas**
Despliegue	• Contratación de host y nombre de dominio. • Despliegue del sistema en modo beta.
Milestone 4	• Entregar dosier del TFG. • Reunión a modo de seguimiento con el tutor.
Milestone 5	• Entregar el artículo.

Milestone 6	• Reunion a modo de seguimiento con el tutor. • Confeccionar presentación en Power Point. • Reunion a modo de seguimiento con el tutor. • Realizar defensa pública.

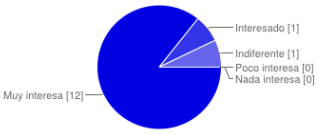
A2. ENCUESTAS

14 respuestas

Ver todas las respuestas Publicar datos de análisis

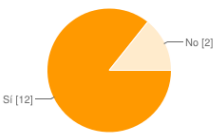
Resumen

1- Desearía que se potenciase el consumo a nivel local?



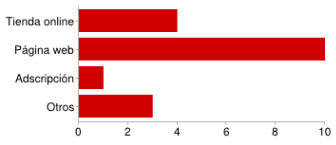
Muy interesado	12	86%
Interesado	1	7%
Indiferente	1	7%
Poco interesado	0	0%
Nada interesado	0	0%

2- Su establecimiento cuenta con presencia online?



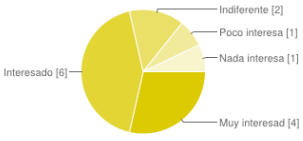
Sí	12	86%
No	2	14%

3- En caso positivo, cual?



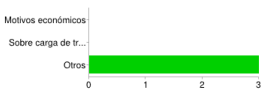
Tienda online	4	29%
Página web	10	71%
Adscripción	1	7%
Otros	3	21%

4- Esta interesado en que su establecimiento disponga de presencia online?



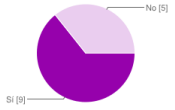
Muy interesado	4	29%
Interesado	6	43%
Indiferente	2	14%
Poco interesado	1	7%
Nada interesado	1	7%

5- En caso negativo, porque?



Motivos económicos	0	0%
Sobre carga de trabajo	0	0%
Otros	3	21%

6- Actualmente cuenta con una base de datos informatizada de su stock?

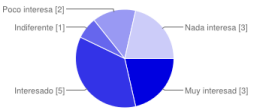


Sí	9	64%
No	5	36%

7- En caso positivo, qué sistema de base de datos o gestor utiliza?

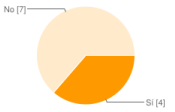
Una aplicación de gestión de librerías
classic gest
ICG MANAGER
excel
no lo saben
PROGRAMA HECHO A MEDIDA
excel y programa de optica

8- Estaría interesado en fomentar el transporte a domicilio a nivel local para fomentar la captación de nuevos clientes?



Muy interesado	3	21%
Interesado	5	36%
Indiferente	1	7%
Poco interesado	2	14%
Nada interesado	3	21%

9- En caso positivo, de manera gratuita?

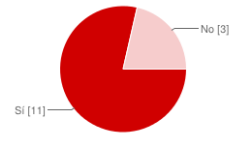


Sí	4	29%
No	7	50%

5- En caso negativo, porque?

Motivos económicos	0	0%
Sobre carga de trabajo	0	0%

10- Está interesado en seguir la evolución del proyecto?



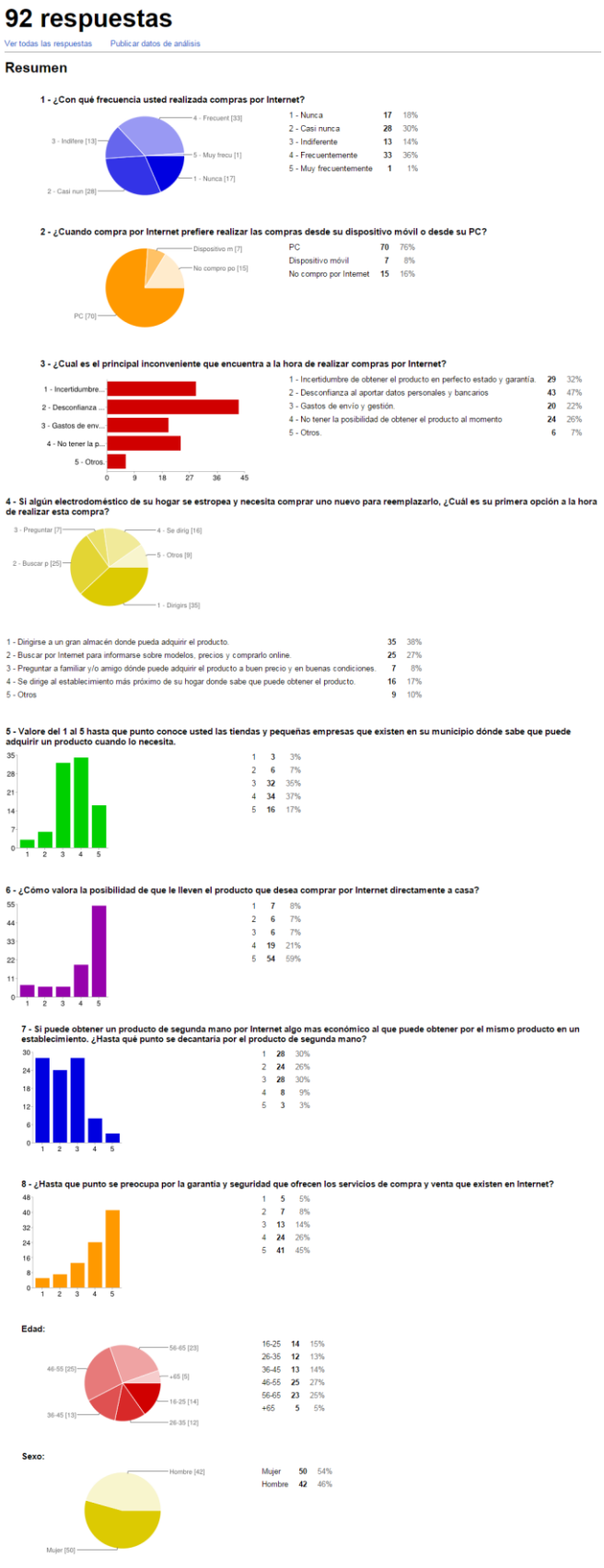
Sí	11	79%
No	3	21%

Datos de contacto:

Ferretería trafalgar
LIMIT SURF SHOP
Forn del passeig forndelpasseig@forndelpasseig.com
Option1-one-inc@hotmail.con
Ventas@stefany-disfraces.com
Zulex, raulpamnani@hotmail.com
DIMESA CORREO ELECTRONICO

Localidad:

barcelona
SANT BOI DE LLOBREGAT
Mollet del Vallès
Barcelona
Palau solità i plegamans
08349 CABRERA DE MAR
mollet del valles barcelona



A3. REQUISITOS

Funcionales:

- RF 1.1: Tan pronto como el usuario acceda a la aplicación, el sistema tiene que mostrar al usuario un listado de productos, servicios y ofertas teniendo en cuenta su proximidad. Prioridad: Alta. Riesgo: Alto
- RF 1.2: El sistema tiene que mostrar los productos, servicios y ofertas situados en un mapa. Prioridad: Alta. Riesgo: Medio.
- RF 1.3: El sistema tiene que proveer al usuario con la habilidad de filtrar y ordenar productos, servicios y ofertas por cercanía precio, puntuación, etc. Prioridad: Media. Riesgo: Medio.
- RF 1.4: El sistema tiene que proveer al usuario con la habilidad de buscar productos, servicios y ofertas por nombre. Prioridad: Media. Riesgo: Alto.
- RF 1.5: El sistema tiene que proveer al usuario con la habilidad de puntuar productos, servicios y ofertas. Prioridad: Baja. Riesgo: Bajo.
- RF 1.6: El sistema tiene que proveer al usuario con la habilidad de consultar la ruta óptima desde su posición al establecimiento deseado. Prioridad: Alta. Riesgo: Alto.
- RF 1.7: El sistema tiene que proveer al usuario con la habilidad de consultar la información relativa a un anunciante. Prioridad: Alta. Riesgo: Alto.
- RF 1.8: El sistema tiene que ser capaz de calcular la localización geográfica del usuario, siempre y cuando el usuario haya aceptado previamente. Prioridad: Alta. Riesgo: Alto.
- RF 1.9: El sistema tiene que proveer al usuario con la habilidad de registrarse en el mismo como usuario consumidor o usuario empresa. Prioridad: Alta. Riesgo: Alto.
- RF: 1.10: El sistema tiene que proveer al usuario con la habilidad de eliminar su cuenta y todos sus datos. Prioridad: Meda. Riesgo: Medio.
- RF 1.11: El sistema tiene que realizar recomendaciones al usuario. Prioridad: Baja. Riesgo: Medio.
- RF 1.12: El sistema tiene que proveer al usuario con la habilidad de contactar con administración. Prioridad: Media. Riesgo: Bajo.
- RF 1.13: El sistema tiene que proveer al usuario con la habilidad de realizar pedidos o reservas. Prioridad: Alta. Riesgo: Alto.
- RF 2.1: El sistema tiene que proveer al usuario empresa con la habilidad de introducir información relativa a su local/comercio como nombre, descripción, horarios, localización. Prioridad: Alta. Riesgo: Alto.
- RF 2.2: El sistema tiene que proveer al usuario empresa con la habilidad de gestionar su catálogo de productos servicios y ofertas. Prioridad: Alta. Riesgo: Alto.
- RF 2.3: El sistema tiene que proveer al usuario empresa con la habilidad de consultar las puntuaciones de los usuarios. Prioridad: Media. Riesgo: Bajo.

- RF 2.4: El sistema tiene que proveer al usuario empresa con la habilidad de gestionar incidencias y pedidos. Prioridad: Alta. Riesgo: Alto.
- RF 2.5: El sistema tiene que proveer al usuario empresa con la habilidad de eliminar su cuenta y todos sus datos. Prioridad: Media. Riesgo: Bajo.
- RF 2.6: El sistema tiene que proveer al usuario empresa con la opción de sincronización automática con su base de datos actual. Prioridad: Baja. Riesgo: Medio.
- RF 2.7: El sistema tiene que proveer al usuario empresa con la habilidad de mejorar su tipo de cuenta. Prioridad: Baja. Riesgo: Medio.

No funcionales:

- RNF1: La web y base de datos deben estar operativas en cualquier momento, permitiendo al usuario poder interaccionar con el sistema sin esperar más de 10 segundos para cualquier operación (siempre y cuando la conexión de red y el proveedor de Internet lo permitan). Prioridad Alta. Riesgo: Alto.
- RNF2: Todos los datos personales sensibles, así como las contraseñas, deberán guardarse cifrados en la base de datos y nunca deberán enviarse en claro a través de la red. Prioridad: Alta. Riesgo: Alto.
- RNF3: El sistema deberá estar preparado para soportar la carga de datos que supondrá pasar de trabajar con datos para pruebas de desarrollo a datos reales de usuarios. Prioridad: Media. Riesgo: Alto.
- RNF4: Si se accede a la aplicación web a través de un ordenador, tanto la funcionalidad como la vista no deberán variar independientemente del sistema operativo y navegador que se utilicen. Prioridad: Media. Riesgo: Medio.
- RNF5: Si se accede a la aplicación web a través de un Smartphone, la funcionalidad deberá ser la misma, sin embargo se deberá mostrar la vista adaptada a dispositivos móviles, sea cual sea el dispositivo, el sistema operativo y el navegador. Prioridad: Baja. Riesgo: Medio.